REMARKS

Claims 48-63 are pending and at issue. Of the pending claims, all stand rejected as either unpatentable under 35 U.S.C. §102(e) based on Mathur et al. (USPN 6,671,745) or unpatentable under 35 U.S.C. §103 based on Mathur et al. in combination with one or more of Brunner et al. (USPN 4,727,544), Pascal (USPN 5,791,851), Angelo (USPN 5,944,821), and a suggested official notice of the Examiner. Of these claims, claims 48 and 61 are independent, and both have been amended above. As discussed in the remarks to follow, applicants respectfully assert that none of the prior art teaches the now recited subject matter.

Provisional Double Patenting Rejection

Applicants note the provisional double patenting rejection based on co-pending U.S. Application Serial No. 09/520,405. Because the rejection is provisional, as there are no issued or as of yet allowed conflicting claims, applicants do not respond to the rejection herein.

Prior Art Rejections

Claim 48 has been amended to recite subject matter similar to that of claim 53. In particular, claim 48 now recites "wherein changing the data variables in the nonvolatile storage causes execution of a corresponding callback function in one of the plurality of program shared objects of the system handler application."

The examiner has rejected claim 53 as obvious over Mathur et al. in combination with Brunner et al. and Pascal. The office action concedes that neither Mathur et al. nor Brunner et al. teach execution of a callback function in response to changing of data variables in a nonvolatile memory. Instead, the office action describes Pascal as teaching a gaming device "wherein callbacks are employed to communicate information between application modules upon the occurrence of certain events." Office Action, page 7. The applicants respectfully disagree that

Pascal teaches the recitations of prior claim 53, now added, at least in part, to claim 48.[1]

Pascal describes an operating system having multiple subsystems each for performing a distinct function and which register with other subsystems so that they can be properly notified (through a callback) upon the occurrence of some event. The Pascal system for example allows subsystems to complete their tasks when a fault is identified and communicated to the subsystem, without commencing further additional activities.

There is no suggestion in Pascal, however, of the claimed subject matter. There is no suggestion of the recited system handler application, which in addition to being operable to initiate a game in response to stored game data variables and that is able to execute certain callback functions in response to changes to data variables stored in a non-volatile storage. Indeed, in reference to Figures 3A-3E, Pascal describes that there are two ways of launching an application, a normal way in which the entire application is loaded at once allowing for multiple parallel applications and an interrupting way in which any currently executing application is first paused and then a second, different application is executed. The office action characterizes Pascal as using callback routines that communicate data, but there is no teaching or suggestion identified by the office action for an operating system "wherein changing the data variables in the nonvolatile storage causes execution of a corresponding callback function in one of the plurality of program shared objects of the system handler application," as recited in claim 48. Sharing data between callback routines does not necessitate that changes in stored data variables will cause execution of corresponding callback functions. Furthermore, the office action points to nothing in Pascal as teaching the execution of a callback function in response to changes in data variables in a non-volatile storage.

In contrast to the teachings of Pascal, the present application allows for selective execution of different call back functions depending on the events occurring

[1] Claim 53 had been dependent from base claims 51 and 52, prior to amendment, but the recitations of these claims have not been added to claim 48.

during runtime, such as via a change in stored game data variables that may occur during a game executing on the operating system. In this way, the entire game need not be loaded at once for execution, but rather only the currently used game code need execute. Furthermore, in this way data dependent code, such as credits to be displayed during a game, may be called by the operating system in response to changes in the data, instead of for example through pre-determined polling of the system for the current state of the data variables.

In short, none of the prior art whether taken alone or in combination teaches the recitations of claim 48. For this reason alone, the rejections of claim 48 and dependent claims 49-52 and 54-60 are traversed.

With respect to independent claim 61, the office action points to Mathur et al. as anticipating the claim. The office action points to discussions in Mathur et al. regarding the interaction between an application and a core system interface 220 and the device manager 210 and device drivers. Yet, the office action appears to consider only where the examiner believes that the system handler application recitations in the claim are taught by Mathur et al. The office action appears to ignore other recitations from the claims, namely those directed to the ability of both the system handler application and the operating system kernel to link and load gaming program shared objects and device handlers. The office action makes no mention of where it is believed that such subject matter is taught by Mathur et al.

The kernel (214) in Mathur et al. is distinctly different from the core system interface (220) and the device manager 210, and there is no suggestion that both a system handler application and a low level operating system kernel can load and execute shared objects and device handlers. In fact, Mathur et al. appears to specifically and exclusively rely on the core system interface 220 to interface with the applications,

> The core system interface 220 is the module through which applications can access the operating system. The core system interface 220 includes functions to transfer

> API calls to the appropriate operating system server process. Mathur et al. 6:47-51.

The device manager module 210 is used to similarly control device drivers:

> Device Manager module 210 is a module that handles installable device drivers. Mathur et al. 9:49-50.

The office action has pointed to no teaching or suggestion of loading and executing directly from both a system handler application and the operating system kernel, and there would appear to be no such teaching in Mathur et al. or anywhere else. As such, the rejection of claim 61 is improper on its face.

The rejection of claim 61 and those of claims 62 and 63 depending therefrom are traversed. Reconsideration and allowance are respectfully requested.

In view of the above amendment, applicants respectfully assert that the pending application is in condition for allowance.

In view of the above amendment, applicant believes the pending application is in condition for allowance.

Dated: May 19, 2006

Respectfully submitted,

By _____

Paul B. Stephens
Registration No.: 47,970
MARSHALL, GERSTEIN & BORUN LLP
233 S. Wacker Drive, Suite 6300
Sears Tower
Chicago, Illinois 60606-6357
(312) 474-6300
Attorney for Applicant